Mixed-Precision GPU-Multigrid Solvers with Strong Smoothers and Applications in CFD and CSM

Dominik Göddeke

Institut für Angewandte Mathematik (LS3)
TU Dortmund
dominik.goeddeke@math.tu-dortmund.de

SIMTECH 2011
International Conference on Simulation Technology
University of Stuttgart, June 16 2011

technische universität dortmund
fakultät für mathematik

Motivation

Hardware isn't our friend any more

- Paradigm shift towards parallelism and heterogeneity
  - In a single chip: Multicores, GPUs, ...
  - In a workstation, cluster node, ...
  - In a big cluster, supercomputer, ...
- Data movement cost gets prohibitively expensive
- Technical reason: Power wall + memory wall + ILP wall = brick wall

Challenges in numerical HPC

- Existing codes don't run faster automatically any more
- Compilers can't solve these problems, libraries are limited
- Traditional numerics is often contrary to these hardware trends
- *We (the numerics people) have to take action*

Hardware-oriented numerics

Conflicting situations

- Existing methods no longer hardware-compatible
- Neither want less numerical efficiency, nor less hardware efficiency

Challenge: New algorithmic way of thinking

- Balance these conflicting goals

Consider short-term hardware details in actual implementations, but long-term hardware trends in the design of numerical schemes

- Locality, locality, locality
- Commmunication-avoiding (-delaying) algorithms between all flavours of parallelism
- Multilevel methods, hardware-aware preconditioning

Grid and Matrix Structures

Flexibility $\leftrightarrow$ Performance

Grid and matrix structures

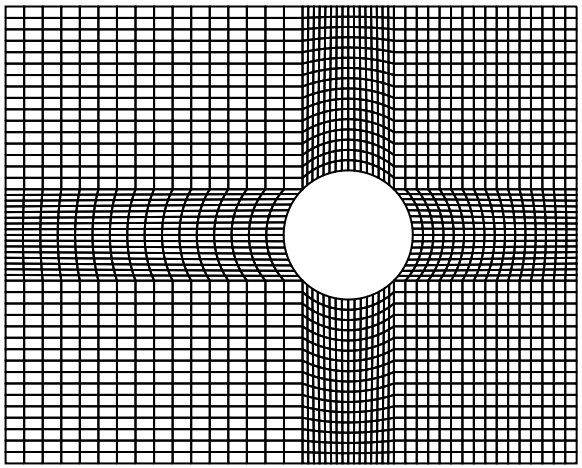
General sparse matrices (unstructured grids)

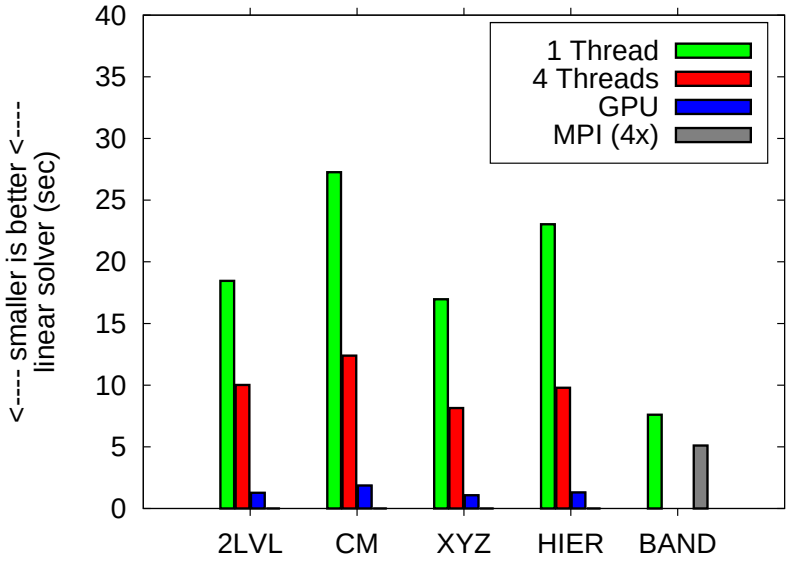
- CSR (and variants): General data structure for arbitrary grids
- Maximum flexibility, but during SpMV
  - Indirect, irregular memory accesses
  - Index overhead reduces already low arithm. intensity further
- Performance depends on nonzero pattern (grid numbering)

Structured sparse matrices

- Example: Structured grids, suitable numbering $\Rightarrow$ band matrices
- Important: No stencils, fully variable coefficients
- Direct regular memory accesses, fast independent of mesh
- Exploitation in the design of strong MG components

Example: Poisson on unstructured mesh





- Nehalem vs. GT200, $\approx$ 2M bilinear FE, MG-JAC solver
- Unstructured formats highly numbering-dependent
- Multicore 2–3x over singlecore, GPU 8–12x over multicore
- Banded format (here: 8 'blocks') 2–3x faster than best unstructured layout and predictably on par with multicore

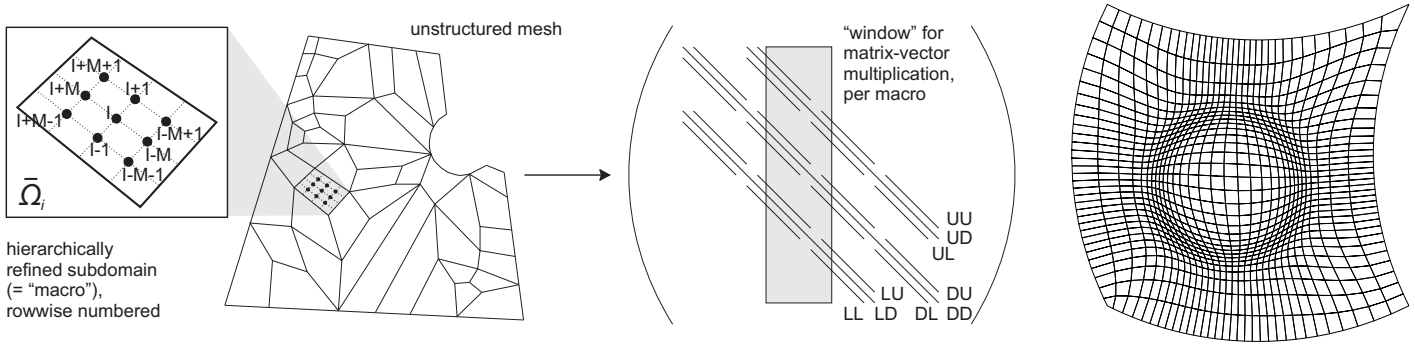
Discretisation and Solver Structures in FEAST

Scalable, Locality-preserving Parallel Multilevel Solvers

Approach in FEAST

Combination of structured and unstructured advantages

- Global macro-mesh: Unstructured, flexible, complex domains
- Local micro-meshes: Structured (logical TP-structure), fast
- Important: Structured $\neq$ simple meshes!



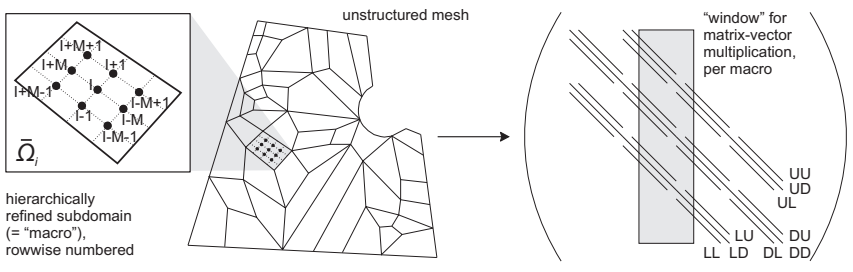
Solver approach ScaRC exploits data layout

- Parallel efficiency: Strong and weak scalability
- Numerical scalability: Convergence rates independent of problem size and partitioning (multigrid!)
- Robustness: Mesh and operator anisotropies (strong smoothers!)

ScaRC: Concepts

ScaRC for scalar systems

- Hybrid multilevel domain decomposition method
- Minimal overlap by extended Dirichlet BCs
- Inspired by parallel MG ('best of both worlds')
  - Multiplicative between levels, global coarse grid problem (MG-like)
  - Additive horizontally: block-Jacobi / Schwarz smoother (DD-like)
- Schwarz smoother encapsulates local irregularities
  - Robust and fast multigrid ('gain a digit'), strong smoothers
  - Maximum exploitation of local structure



global BiCGStab

preconditioned by

global multilevel (V 1+1)

additively smoothed by

for all Ω_i: **local multigrid**

coarse grid solver: UMFPACK

ScaRC for multivariate problems

Block-structured systems

- Guiding idea: Tune scalar case once per architecture instead of over and over again per application
- Blocks correspond to scalar subequations, coupling via special preconditioners
- Block-wise treatment enables *multivariate ScaRC solvers*

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \mathbf{f},$$

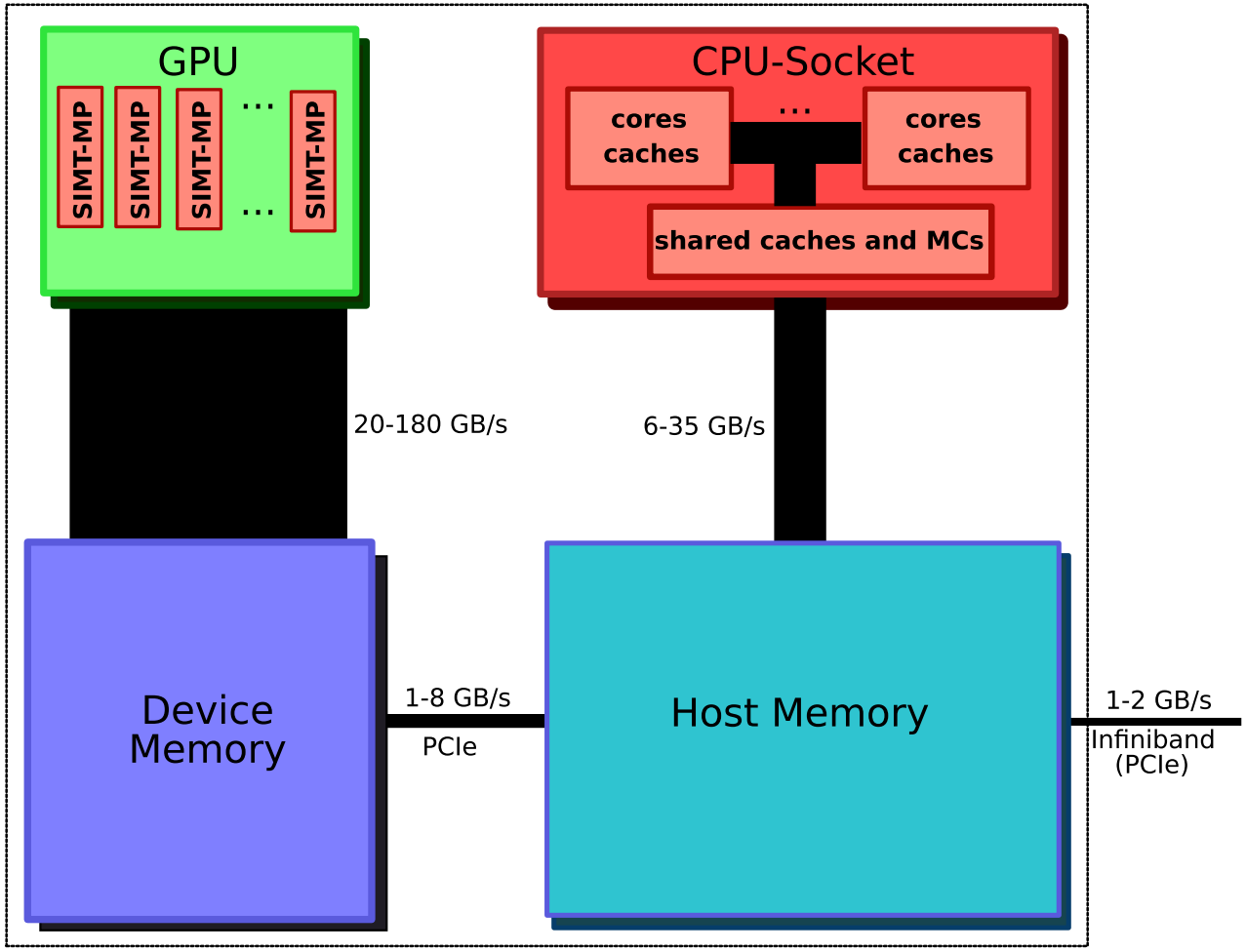
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{B}_1 \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{B}_2 \\ \mathbf{B}_1^T & \mathbf{B}_2^T & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{p} \end{pmatrix} = \mathbf{f}, \quad \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{B}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{B}_2 \\ \mathbf{B}_1^T & \mathbf{B}_2^T & \mathbf{C}_C \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{p} \end{pmatrix} = \mathbf{f}$$

$\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ correspond to scalar (elliptic) operators

$\Rightarrow$ Tuned linear algebra **and** tuned solvers

Minimal invasive accelerator integration

Bandwidth distribution in a hybrid CPU/GPU node

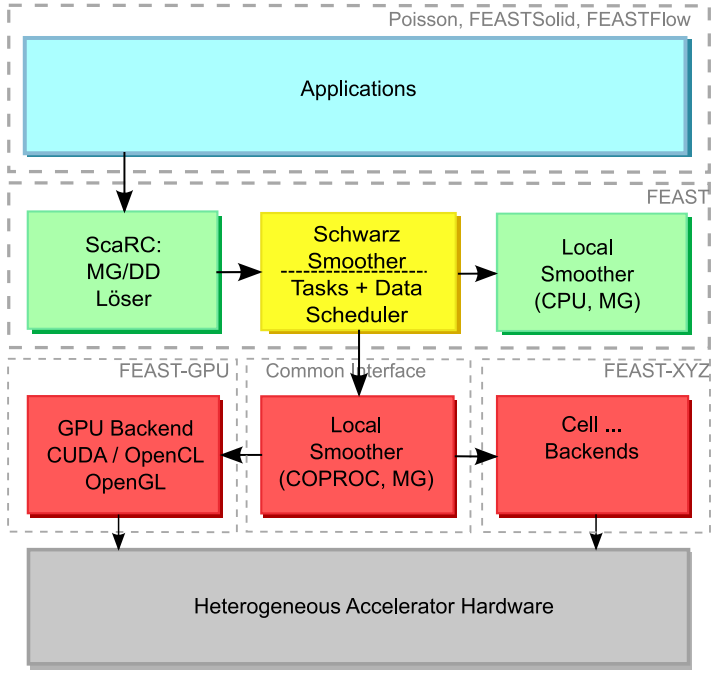


Minimally invasive accelerator integration

Guiding concept: locality

- Accelerators: Most time-consuming inner component
- CPUs: Outer MLDD solver (only hardware capable of MPI anyway)
- Block-structured approach inside MPI rank allows double-buffering and PCIe communication overlap
- Employ mixed precision approach

global BiCGStab
preconditioned by
  global multilevel (V 1+1)
  additively smoothed by
    for all Ω_i: **local multigrid**
  coarse grid solver: UMFPACK



Minimally invasive accelerator integration

Benefits and challenges

- Balance acceleration potential and integration effort
- Accelerate many different applications built on top of one central FE and solver toolkit
- Diverge code paths as late as possible
- Develop on a single GPU and scale out later
- Retain all functionality
- Do not sacrifice accuracy
- No changes to application code!

Challenges

- Heterogeneous task assignment to maximise throughput
- Overlapping CPU and GPU computations with transfers

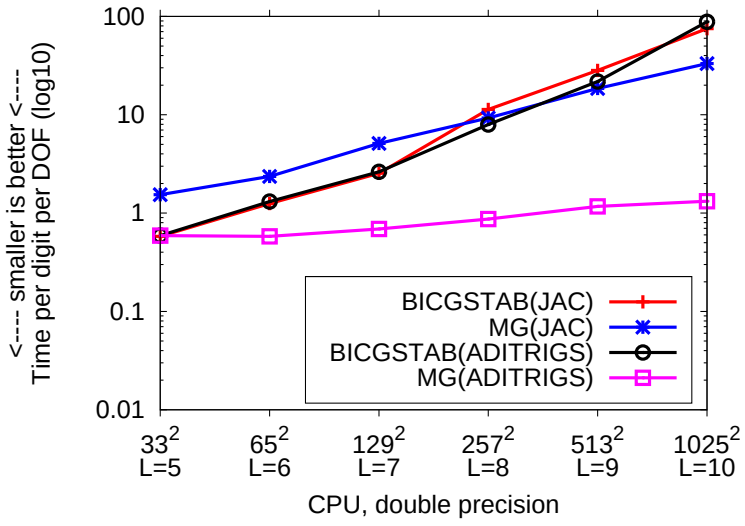
Strong Smoothers

Parallelising Inherently Sequential Operations

Motivation: Why strong smoothers?

Test case: Generalised Poisson problem with anisotropic diffusion

- $-\nabla \cdot (\mathbf{G}\ \nabla \mathbf{u}) = \mathbf{f}$ on unit square (one FEAST patch)
- $\mathbf{G} = \mathbf{I}$: standard Poisson problem, $\mathbf{G} \neq \mathbf{I}$: arbitrarily challenging
- Example: $\mathbf{G}$ introduces anisotropic diffusion along some vector field



Only multigrid with a strong smoother is competitive

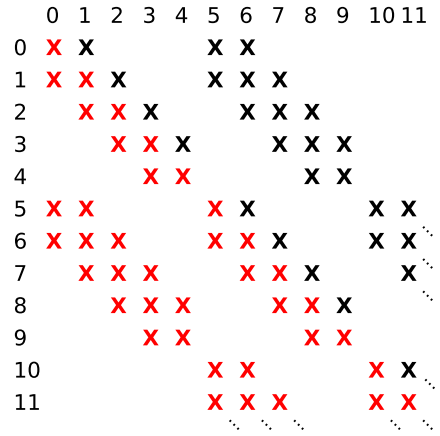
Gauß-Seidel smoother

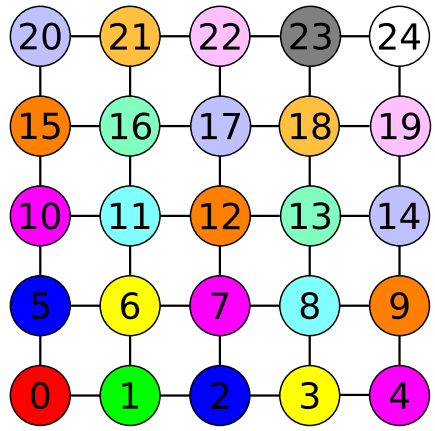
Disclaimer: Not necessarily a good smoother, but a good didactical example.

Sequential algorithm

- Forward elimination, sequential dependencies between matrix rows
- Illustrative: Coupling to the left and bottom

1st idea: Classical wavefront-parallelisation (exact)



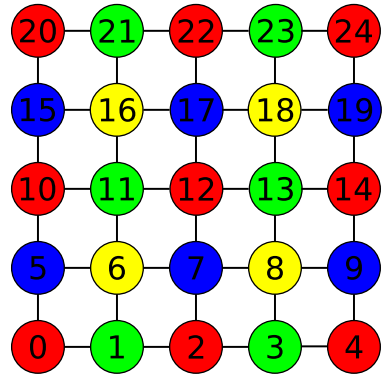


- Pro: Always works to resolve *explicit* dependencies
- Con: Irregular parallelism and access patterns, implementable?

Gauß-Seidel smoother

2nd idea: Decouple dependencies via multicolouring (inexact)

- Jacobi (red) – coupling to left (green) – coupling to bottom (blue) – coupling to left and bottom (yellow)



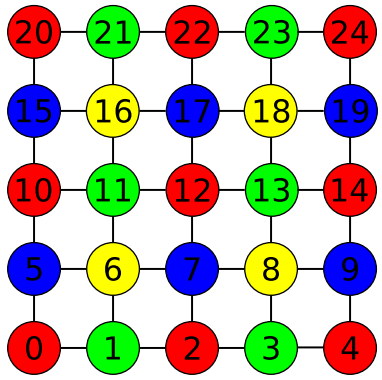
Analysis

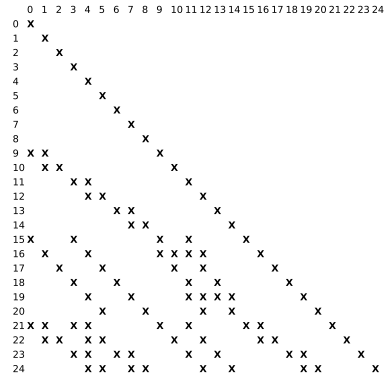
- Parallel efficiency: 4 sweeps with $\approx N/4$ parallel work each
- Regular data access, but checkerboard pattern challenging for SIMD/GPUs due to strided access
- Numerical efficiency: Sequential coupling only in last sweep

Gauß-Seidel smoother

3rd idea: Multicolouring = renumbering

- After decoupling: 'Standard' update (left+bottom) is suboptimal
- Does not include all already available results



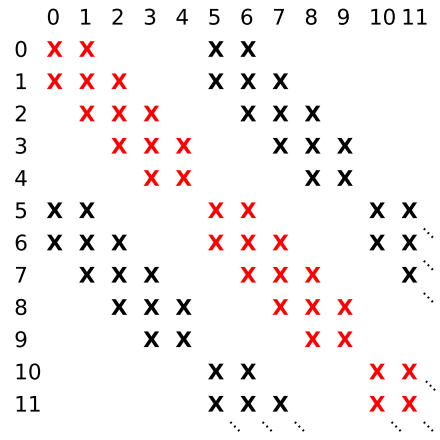


- Recoupling: Jacobi (red) – coupling to left and right (green) – top and bottom (blue) – all 8 neighbours (yellow)
- More computations that standard decoupling
- Experiments: Convergence rates of sequential variant recovered (in absence of preferred direction)

Tridiagonal smoother (line relaxation)

Starting point

- Good for 'line-wise' anisotropies
- *'Alternating Direction Implicit (ADI)'* technique alternates rows and columns
- CPU implementation: Thomas-Algorithm (inherently sequential)



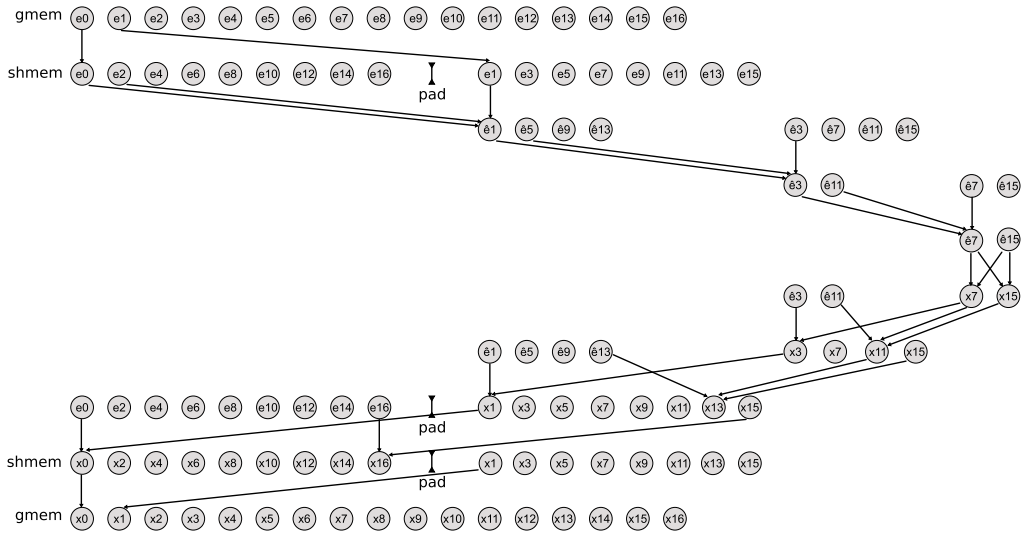
Observations

- One independent tridiagonal system per mesh row
- $\Rightarrow$ top-level parallelisation across mesh rows
- Implicit coupling: Wavefront and colouring techniques not applicable

Tridiagonal smoother (line relaxation)

Cyclic reduction for tridiagonal systems

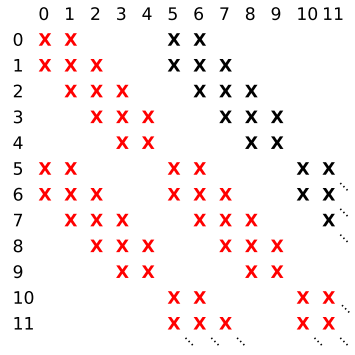
- Exact, stable (w/o pivoting) and cost-efficient
- Problem: Classical formulation parallelises computation but not memory accesses on GPUs (bank conflicts in shared memory)
- Developed a better formulation, 2-4x faster
- Index challenge, general idea: Recursive padding between odd and even indices on all levels



Combined GS and TRIDI

Starting point

- CPU implementation: Shift previous row to RHS and solve remaining tridiagonal system with Thomas-Algorithm
- Combined with ADI, this is the best general smoother (we know) for this matrix structure



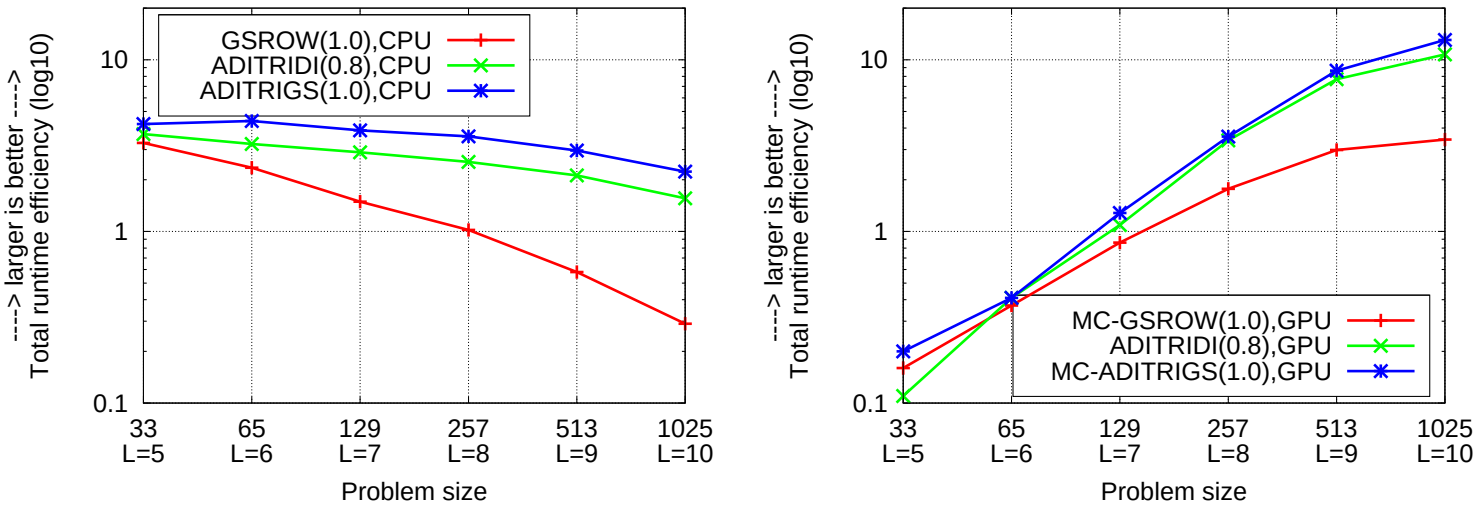
Observations and implementation

- Difference to tridiagonal solvers: Mesh rows depend sequentially on each other
- Use colouring ($\#c \geq 2$) to decouple the dependencies between rows (more colours = more similar to sequential variant)

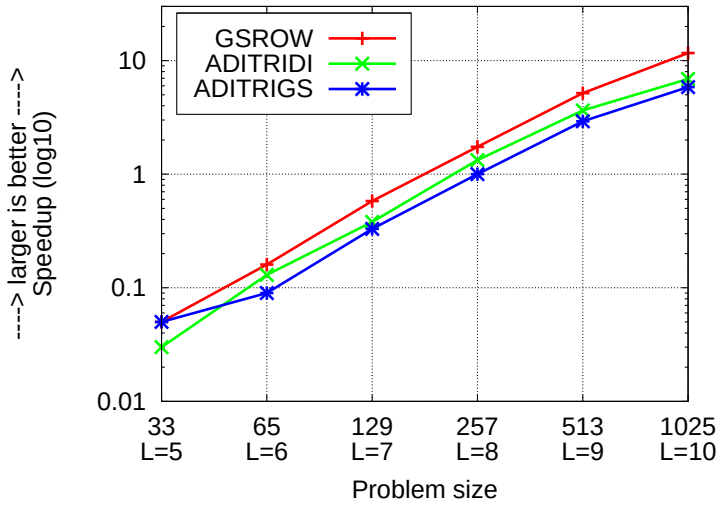
Evaluation: Total efficiency on CPU and GPU

Test problem: Generalised Poisson with anisotropic diffusion

- Total efficiency: Time per unknown per digit (μs)
- Mixed precision iterative refinement multigrid solver
- Intel Westmere vs. NVIDIA Fermi



Speedup GPU vs. CPU



Summary: Smoother parallelisation

- Factor 10-30 (dep. on precision and smoother selection) speedup over already highly tuned CPU implementation
- Same functionality on CPU and GPU
- Balancing of numerical and parallel efficiency (hardware-oriented numerics)

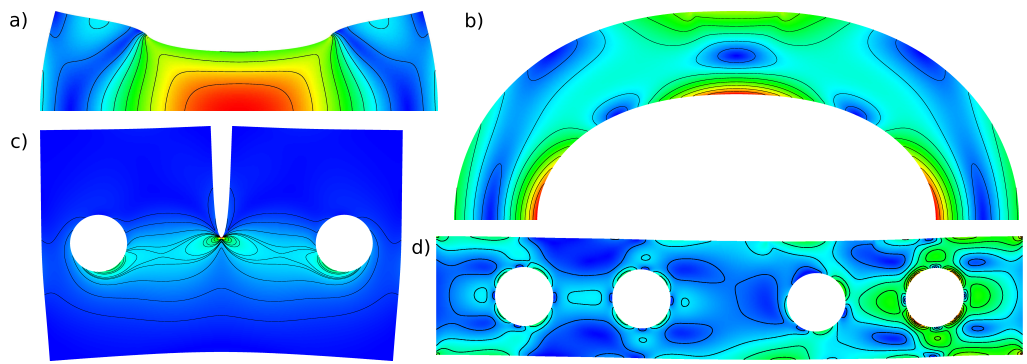
Cluster Results

Linearised elasticity

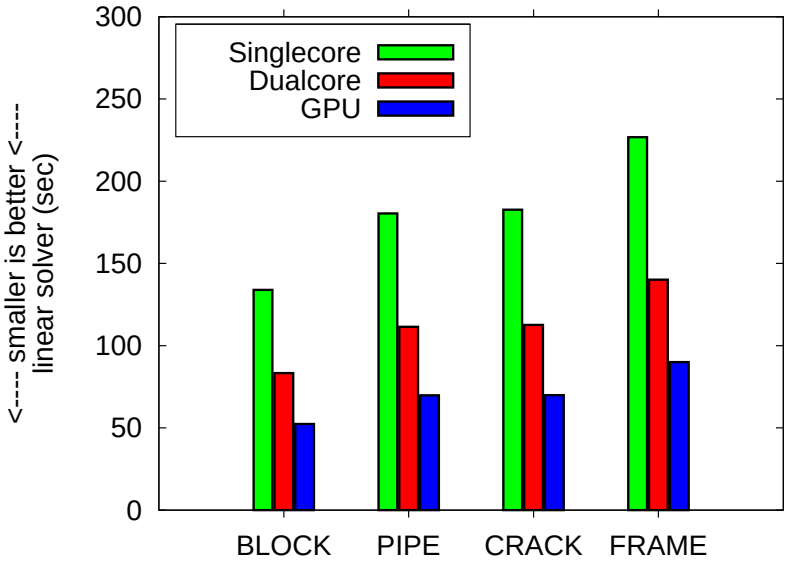
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \mathbf{f}$$

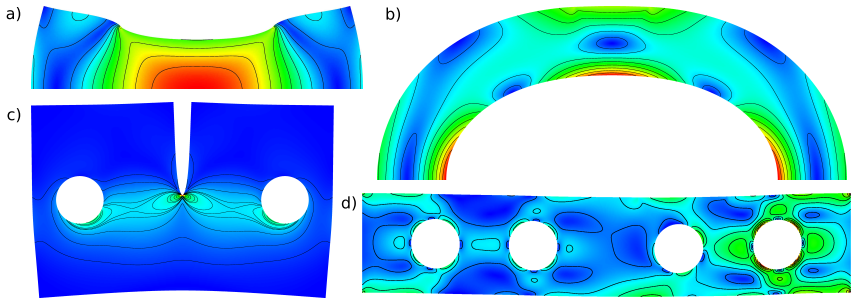
$$\begin{pmatrix} (2\mu+\lambda)\partial_{xx} + \mu\partial_{yy} & (\mu+\lambda)\partial_{xy} \\ (\mu+\lambda)\partial_{yx} & \mu\partial_{xx} + (2\mu+\lambda)\partial_{yy} \end{pmatrix}$$

global multivariate BiCGStab
block-preconditioned by
Global multivariate multilevel (V 1+1)
additively smoothed (block GS) by
for all Ω_i: solve $\mathbf{A}_{11}\mathbf{c}_1 = \mathbf{d}_1$ by
local scalar multigrid
update RHS: $\mathbf{d}_2 = \mathbf{d}_2 - \mathbf{A}_{21}\mathbf{c}_1$
for all Ω_i: solve $\mathbf{A}_{22}\mathbf{c}_2 = \mathbf{d}_2$ by
local scalar multigrid
coarse grid solver: UMFPACK



Speedup





- USC cluster in Los Alamos, 16 dualcore nodes (Opteron Santa Rosa, Quadro FX5600)
- Problem size 128 M DOF
- Dualcore 1.6x faster than singlecore (memory wall)
- GPU 2.6x faster than singlecore, 1.6x than dualcore

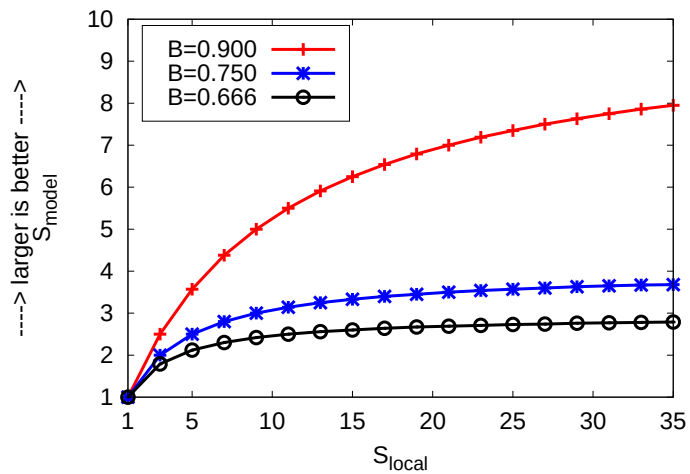
Speedup analysis

Theoretical model of expected speedup

- Integration of GPUs increases resources
- Correct model: Strong scaling within each node
- Acceleration potential of the elasticity solver: $R_{\text{acc}} = 2/3$ (remaining time in MPI and the outer solver)
- $S_{\text{max}} = \frac{1}{1-R_{\text{acc}}}$ $\qquad$ $S_{\text{model}} = \frac{1}{(1-R_{\text{acc}})+(R_{\text{acc}}/S_{\text{local}})}$

This example

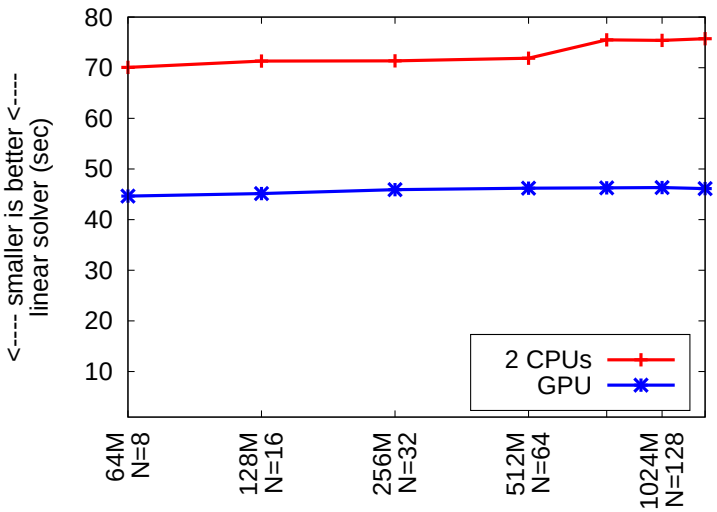
Accelerable fraction R_{acc}	66%
Local speedup S_{local}	9x
Modeled speedup S_{model}	2.5x
Measured speedup S_{total}	2.6x
Upper bound S_{max}	3x

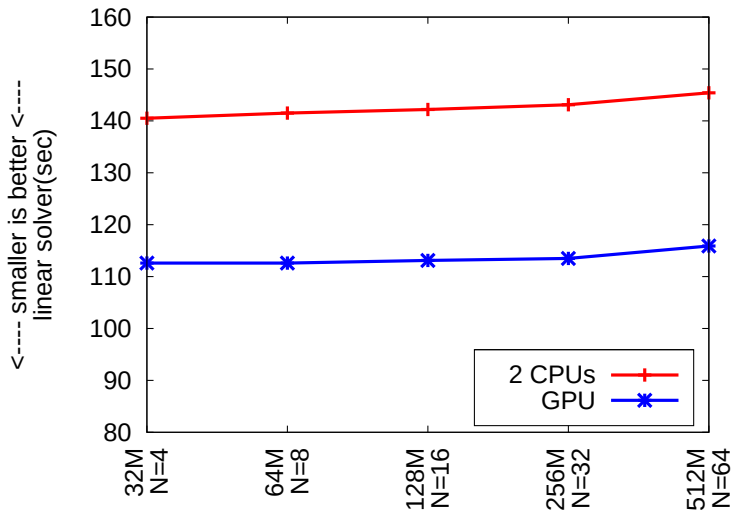


Weak scalability

Simultaneous doubling of problem size and resources

- Left: Poisson, 160 dual Xeon / FX1400 nodes, max. 1.3 B DOF
- Right: Linearised elasticity, 64 nodes, max. 0.5 B DOF





Results

- No loss of weak scalability despite local acceleration
- 1.3 billion unknowns (no stencil!) on 160 GPUs in less than 50 s

Stationary laminar flow (Navier-Stokes)

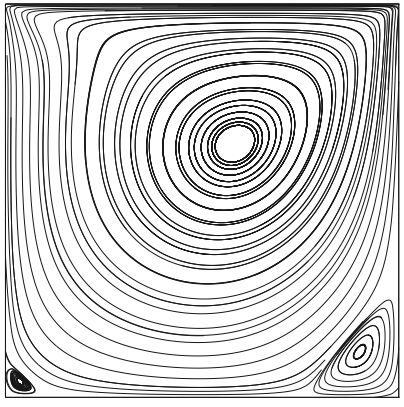
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{B}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{B}_2 \\ \mathbf{B}_1^T & \mathbf{B}_2^T & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{g} \end{pmatrix}$$

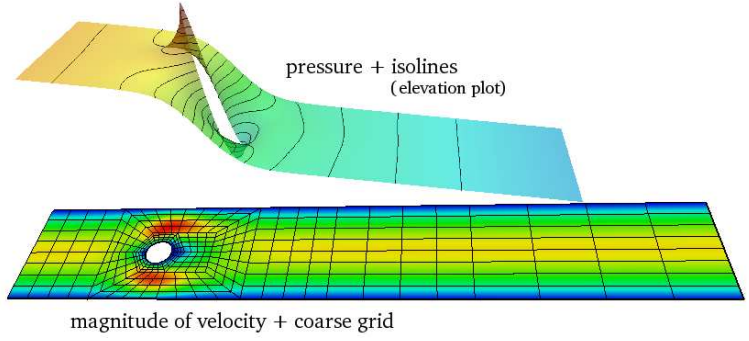
fixed point iteration
assemble linearised subproblems and solve with
global BiCGStab (reduce initial residual by 1 digit)
Block-Schurcomplement preconditioner
1) approx. solve for velocities with
global MG (V 1+0), additively smoothed by

for all Ω_i: solve for $\mathbf{u}_1$ with
local MG

for all Ω_i: solve for $\mathbf{u}_2$ with
local MG

2) update RHS: $\mathbf{d}_3 = -\mathbf{d}_3 + \mathbf{B}^T(\mathbf{c}_1, \mathbf{c}_2)^T$
3) scale $\mathbf{c}_3 = (\mathbf{M}_p^L)^{-1}\mathbf{d}_3$





Stationary laminar flow (Navier-Stokes)

Solver configuration

- Driven cavity: Jacobi smoother sufficient
- Channel flow: ADI-TRIDI smoother required

Speedup analysis

	R_{acc}		S_{local}		S_{total}	
	L9	L10	L9	L10	L9	L10
DC Re250	52%	62%	9.1x	24.5x	1.63x	2.71x
Channel flow	48%	–	12.5x	–	1.76x	–

Shift away from domination by linear solver

- Fraction of FE assembly and linear solver of total time, max. problem size

DC Re250		Channel	
CPU	GPU	CPU	GPU
12:88	31:67	38:59	**68:28**

Summary

Summary

ScaRC solver scheme

- Globally-unstructured-locally-structured
- Tight co-design of discretisation (grid and finite elements) with multilevel solver
- Beneficial on CPUs and GPUs
- Numerically and computationally future-proof (some odd ends still to be resolved)

GPU computing

- Parallelising strong recursive smoothers
- Minimally invasive acceleration with legacy codes

Significant speedups

- On a single device: one order of magnitude
- On the application level: Reduced due to Amdahl's Law

Acknowledgements

Collaborative work with

- FEAST group (TU Dortmund): Ch. Becker, S.H.M. Buijssen, M. Geveler, D. Göddeke, M. Köster, D. Ribbrock, Th. Rohkämper, S. Turek, H. Wobker, P. Zajac
- Robert Strzodka (Max Planck Institut Informatik)
- Jamaludin Mohd-Yusof, Patrick McCormick (Los Alamos National Laboratory)

Supported by

- DFG: TU 102/22-1, TU 102/22-2
- BMBF: *HPC Software für skalierbare Parallelrechner*: SKALB project 01IH08003D

http://www.mathematik.tu-dortmund.de/~goeddeke